CHAPTER

2

IT Impasse

Faster Software Development Versus Operational Stability

Ever since we flipped the switch on commercial computers back in the 1950s, IT departments have been struggling to keep up with an insatiable demand for software applications and services. Of course many technologies like commercial of-the-shelf software packages, virtualization, and cloud computing have helped along the way, but generally IT delivery has been slow and uncoordinated.

In such an environment where IT generally supported internally-centric business processes, separate teams overseeing discrete technology functions was for many years considered the norm. But in today's rapidly evolving digital world, these practices no longer work.

A World of 'Wicked' Business Problems

Keeping up with demand for changes to internal applications supporting business processes is tame compared to the "wicked" problems facing IT departments today. Like societal problems such as global warming and drug abuse, they're wicked because IT is placed in an unenviable position of trying

A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_2

to rapidly deliver solutions before problems are fully understood and where business conditions constantly change. This is due to three transformational forces:

- *Products to services*—Customers now care less about physical things and more about the total experience. This explains why companies wrap physical products in services. Like Tesla, who routinely deliver enhancements to the Model S car via software. Or Bosch, who now provides tailored telematics and analytics as-a-services so that fleet operators can optimize maintenance and cut fuel costs.
- *Efficiency to agility*—Established brands with decades of reputation are constantly being disrupted. Kodak lasted 100 years, Blockbuster less than 20. Even technology stalwarts like Microsoft have felt the ground shift beneath them. Business reputations are forged from digital adeptness and the ability stay ahead of the market—or create new ones.
- *Separation to fusion*—There is no longer separation between physical products and software. Is your smartphone circuitry and plastic, or is it a Spotify music service and digital payment system? Is your Nest home thermostat an aesthetically pleasing appliance, or is it an analytical marvel of energy management?

Operating within this environment, wicked problems now challenge the essence of how business is conducted and how applications should be designed, developed, and supported. Thanks to mobility and social computing, the producer-consumer relationship has been reversed, with businesses placed in a position of having to respond to customer behaviors rather than dictating them. This places many organizations on the back foot because of traditional development practices.

Internal business processes supported by large complex applications have been the lifeblood of business. Supporting processes like logistics, inventory, and financial management, these applications are periodically optimized to eke out operational cost efficiencies. In this context, the focus on IT has been to maintain a steady state; keeping the technical lights and only changing applications over longer cycles, sometimes only once or twice a year.

Even when custom development occurs, the general practice has been to invest heavily in application software to support large-scale projects. Here, the pattern is to define all the requirements and features at the start and progress the application toward production status by developing and testing in a linear fashion. Finally, if many business stakeholders agree the system is what's wanted, the application is handed over the wall to production for IT operations to maintain.

Considering these forces, this "Waterfall" style of development (see Figure 2-1) now has a number of disadvantages. First, by establishing a fully defined set of requirements up-front and then failing to accommodate shifting customer behaviors, organizations risk delivering "white elephant" applications, which are software systems that customers never use. Secondly, in the time taken to release software, business conditions may have changed, meaning departments must change the system radically, or as is often the case, abandon it completely. Finally, since application software and functionality is delivered en masse, the support and maintenance burden on IT operations increases suddenly and significantly.

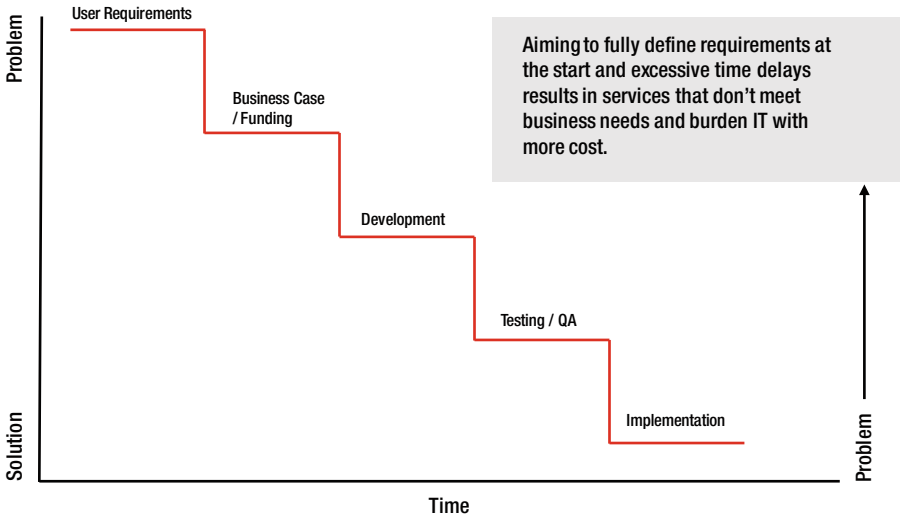

Figure 2-1. Waterfall software development model

The Emergence of Agile Development

In Formula 1 racing, rules change every season. Cars now compete without fuel pit-stops and on sets of tires that degrade more quickly. It's the same in IT, where software must now be delivered as a continuous stream of value that constantly changes to support new business conditions.

Trying to solve wicked business problems (e.g., transforming the businesses operational model with a new mobile sales channel) with existing methodologies like Waterfall can be like trying to corral cats. Problem definitions and requirements can change as new solutions are considered and implemented, so much more flexibility is required across the software development lifecycle. And, with many diverse opinions on what actually constitutes a business problem, there'll be many stakeholders to engage and ideas to analyze.

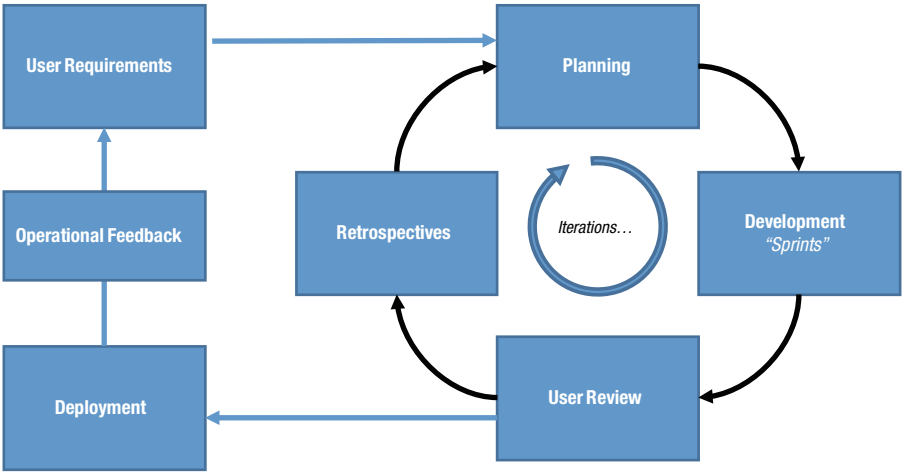

Figure 2-2. Agile software development model

Agile development has emerged as the practice of choice to address these challenges. Agile (see Figure 2-2), which involves iterative problem solving and continuous feedback, is well suited to solving these complex business problems. Some common benefits include:

- Software updates are made in small regular batches, allowing businesses to adapt quickly to new information
- Mistakes or false assumptions are detected much earlier in the software lifecycle, where they are much less costly to correct
- By detecting problems quickly and early, teams have much faster feedback and organizational knowledge improves
- Teams can immediately apply learning and knowledge gained to improve the quality of application software
- By working in parallel across development and testing, agile teams can increase the velocity of application development and delivery

Agile methods help teams manage and run their own projects. Rather than wait for approvals and sign-offs across multiple stages, agile supports the notion that fully autonomous teams should be able to make their own architectural design decisions, even when it comes to tool usage and deciding what's needed to push software code all the way into production.

But agile development is not without its drawbacks. While agile development teams are focused on speed and agility, the traditional mantra of IT operations is maintaining application stability, even if that means slowing things down. To this end, IT operations often employ rigorous change control processes and enforce infrastructure standardization dictates to maintain control. While well suited to supporting legacy internal systems where updates were delivered in large batches, these processes can be too rigid to support newer applications designed to be always changing.

Many of these issues are a direct consequence of how IT operations have traditionally been run. Historically, IT operations have been delivered as a set of shared services, supported by functional disciplines in areas such as data center services, network management, and security. If business users or development needed an enabling service (e.g., provisioning test labs), operations had to be engaged for delivery.

However, this model is changing, with new tools and methods that enable business groups and development teams to bypass the IT operations shared service completely. Many recent technology trends have supported these practices, including:

- *Polyglot programming languages/platforms*—Autonomous teams can select the programming language best suited to their project. This may include older languages like Java and C++, or new platforms and languages to support specific development use-cases or design requirements (e.g., Node.JS, Go, and Rust). In other cases development teams are eschewing traditional relational databases in favor of NoSQL or document style data stores. These could be advantageous in projects where data requirements are indeterminate or evolving and where speed and scalability is more critical than up-front logical design and data integrity. Examples include MongoDB, PostgreSQL, and Cassandra.
- *Programmable Infrastructure*—Also known as Infrastructure-as-code, this allows teams to write code to manage configurations and automate infrastructure provisioning. Rather than using manual methods or scripting to build development ready infrastructure, teams can write code in languages with which they are familiar, together with familiar practices such as version control and automated testing to reduce errors. Unlike scripting, which is primarily used to automate static steps, programmable infrastructure with its descriptive languages allows developers to code processes that are far more versatile and adaptive. Examples include Chef, Puppet, and Ansible.

- *Open source software*—Beyond the more obvious "free to use" benefits, many organizations are embracing open source software (OSS) as a means to increase agility. By adopting OSS, development teams can benefit from community-based and collaborative development, with tools that simplify and automate many tasks (e.g., version control and software build management). By having the eyes of the "community" on the software, bugs can be quickly identified and resolved, with new improvements constantly being developed and delivered.
- *Cloud and platform-as-a-service (PaaS)*—Not too long ago testing a new application or update required development managers to submit a business case for the procurement of new hardware, wait weeks for delivery, then go through a painstaking process of provisioning and configuration. This has changed in recent years—first with the advent of server virtualization and more recently with PaaS providing complete cloud technology stacks for development and testing.
- *Containers*—A recent technology innovation, containers comprise an entire runtime environment, including the application, plus all its dependencies, libraries, and configuration files needed to run it—all bundled into one package. By containerizing the application platform and all dependencies, any differences in OS distributions and underlying infrastructure are abstracted away. Unlike virtual machines, containerized applications run a single operating system, and each container shares the operating system kernel with the other containers. This makes containers more lightweight and resource efficient than virtual machines. Containers can be run on public cloud services and are easily shared, which makes them particularly useful for development and testing teams.
- *Canary releases/dark launches*—This is a release method used to test the performance of software deployments and new functionality in a phased manner. Canary releases are an important aspect of agile development, since, by rolling out new features incrementally and testing them with real users, teams can gather feedback and implement improvements quickly.

- *Design for failure*—As businesses embrace public cloud services, teams are employing design methods to make applications completely independent of the availability of underlying infrastructure. By building resilient systems in which inevitable failures have a minimum impact on service, design for failure allows teams to scale applications quickly while achieving higher levels of uptime, even during major cloud infrastructure outages.

Agile Empowerment Challenges

While many of the technologies and methods described above have empowered teams to deliver software faster, this doesn't guarantee success. While developers are tasked with producing great software code, they often neglect critical issues associated with application performance and supportability. In many cases the adoption of new technologies and practices exacerbates this problem due to some common failings:

Technology for Technologies' sake—It's understandable that developers want to use the latest tool, but this can lead to support problems. For example, opting for a NoSQL database because it helps avoid lengthy schema design issues may address short-term issues, but increases the support burden because IT operations and support have no experience with the technology. Operational cost structures can also be impacted, by way of increased training costs and hiring specialists.

Tip Make new technology tool selection a collaborative exercise. Enterprise architects can coach teams on the importance of placing program or business level objectives above discrete tool requirement or personal preferences.

Misuse of New Technology—It's a fact of IT that new technologies often can and will be abused, unwittingly or intentionally. With modern technologies, lack of experience may entice people to use technologies in ways they were never designed. For example, blindly dictating that every monolithic and legacy application should be containerized whether they're appropriate for the technology or not.

Tip Today's new technology is tomorrow's legacy. Always assess whether the use of technology is appropriate for the business and never underestimate architecture and design requirements.

Metric Misalignment—With developers using techniques like canary releases, dark launches, and split or A/B testing, businesses needs to know whether new features are successful and have led to more customer conversions and sales. Historically, however, IT operations have used technical diagnostics as a means to assess performance, which may be misaligned with the business.

Tip Consider adopting monitoring methods that focus more on achieving desired business outcomes. In a mobile app context, this could include monitoring techniques to analyze app and functional usage by geographic area and user community. By managing to these outcomes, it becomes easier to assess the implications of any application performance issues and feed richer information and insights back to development.

Lost Opportunities—As new development platforms become increasingly abstracted from hardware infrastructure, many complex performance problems can surface. Because developers are shielded from the infrastructure, they can be slow to respond to any issues their code has introduced. This lack of insight also means they can fail to fully exploit the true potential of new cloud-based technologies.

Tip In customer-centric computing, high-performance and low-latency are as important as functionality and design. Consider teaming junior and experienced developers with skilled IT operations and sysadmins to determine how modern hardware architectures can be better exploited to consistently deliver a high-quality customer experience.

Workarounds Due to Constraints—Even as teams look to adopt public cloud services and PaaS to accelerate development, they're still constrained by access to production systems for testing. This can involve delayed access to applications, infrastructure, and perhaps the most time-consuming and resource-intensive task in IT—creating, maintaining, and provisioning accurate, compliant and up-to-date production-like test data. With these constraints, teams may introduce sub-optimal testing practices that fail to detect software defects or compromise compliance with regulatory data protection requirements.

Tip Consider technologies that can simulate unavailable systems across the development lifecycle. This allows developers, testers, and integration teams to work in parallel for faster delivery. Capabilities in test data management, including data subsetting and masking, together with synthetic on-demand test data generation, should also be assessed.

Modern Application Architectures

To support the businesses need to innovate, development teams must continuously deliver software services at an increased velocity. This has been recognized by web-native companies such as Netflix and Amazon, who've changed their software architectures to support the need for continuous innovation—essentially using them to redefine the markets within which they operate.

With agile methods, organizations can iterate quickly to support innovation, but teams are also changing application architectures to improve software flexibility and help accelerate deployment. For this reason, older style monolith designs are being supplemented with microservice designs (see Figure 2-3).

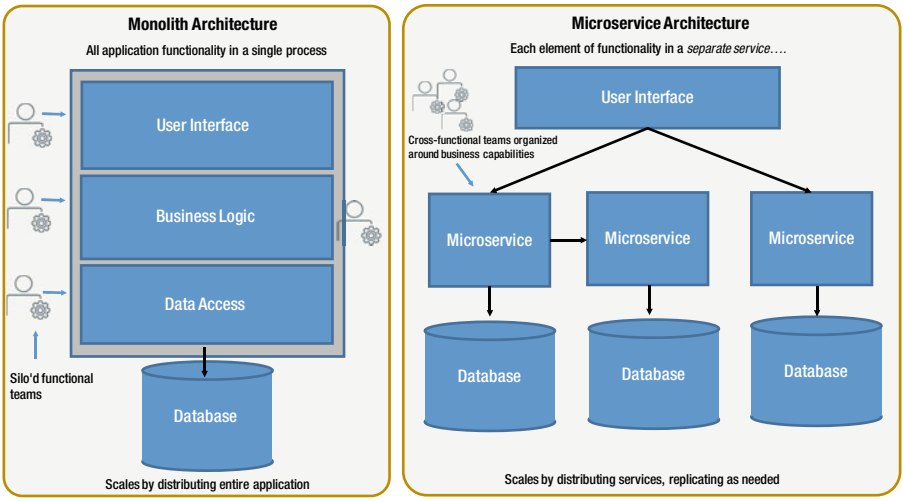

Figure 2-3. Monolithic and microservice application architectures

Until recently, application architectures were monolithic in design and operation. Although consisting of many services, monolithic applications are packaged and operate as a single unit. For IT teams now tasked with faster delivery and deployment, the characteristics of monolithic design have presented a number of operational challenges:

- *Brittleness*—If any single application element of component fails, then the entire application fails. If a task such as payment processing consumes more CPU or memory, then the whole application can degrade.
- *Risk*—Because everything is packaged together (and fails together), teams may be hesitant to change supporting technology stacks and infrastructure. This can explain

operational resistance to increased deployment rates, since even simple application updates to brittle monolithic applications could cause system-level outages.

- *Tightly coupled*—Applications can only be scaled by deploying the entire application on more servers. This can be highly problematic in new mobile application development scenarios when demand is difficult to predict.
- *Dependencies*—Since applications are tied together, developers may have difficulty working independently to develop, test, and deploy their own software components. Development time increases and productivity suffers because they're often dependent on other teams finishing work before they can start.

Microservice designs differ from monoliths in that they involve building applications as a set of small, independent services. In essence, each microservice focuses on a specific element of business functionality. For example, in a web application there could be many microservices supporting everything from login processing and shopping carts, recommendation services and payment processing. Loosely couple in nature, microservices communicate with other services via Application Programming Interfaces (APIs).

Microservices can provide business a number of significant benefits:

- *Independent deployment*—Scaling becomes less problematic. For example, in a web-based shopping application a team can quickly deploy the instances of service it needs to meet demand spikes, while scaling back others.
- *Independent coding*—Teams have more freedom to develop in different programming languages, each optimized for different processing tasks. Microservices can free organizations from being locked into a single technology stack.
- *Fault tolerance*—When a microservice fails, it's unlikely that the entire system will fail. If the recommendation service fails in a web application, shopping cart and payment processing services can continue to function.
- *Increased agility*—Microservices designs can better support continuous delivery. Since systems are built and deployed independently, they can potentially be tested and delivered faster.

Microservices: Small Isn't Always Beautiful

While the simple and elegant nature of microservice style design delivers many benefits, substantial complexity exists in terms of management and operations. If ignored, these issues could increase friction between development and operations teams. Important considerations include:

- *Supportability*—The support burden can increase substantially—especially when IT operations are suddenly faced with managing hundreds of microservices developed in different languages, accessing new data stores, and running on cloud platforms and infrastructure.
- *Monitoring*—Managing a single monolithic application is demanding, but now IT operations has to ensure many more processes remain performant. With highly distributed microservices systems, a whole new set of management considerations surface. These include network latency, fault tolerance, asynchronous messaging issues, and network reliability.
- *Coordination*—Deploying hundreds of microservices demands rigorous deployment processes that can be beyond the capabilities of teams using manual processes and scripting.

Note Modern management approaches to address microservice deployment and monitoring challenges are discussed further in Chapters 6 and 7.

Ending the Technical Impasse

Even with advances in tooling, development methods and software designs, the friction created by teams operating in discrete functional silos can negate all potential benefits. Fractured processes across the application software lifecycle inevitably result in slow delivery, low productivity, and defect-ridden software systems. This has to change.

IT's value proposition isn't just to keep the technology lights on and periodically deliver improvements over long cycles; it's to continuously manufacture business value from a modern high-performance software factory.

DevOps, with its focus on close collaboration between development and other IT groups, while automating essential application delivery processes, is now a critical business imperative.

Summary

The following chapters outline key strategies that can help IT and digital leaders accelerate a successful and business-aligned DevOps initiative.

Starting with Chapter 3, we'll describe how to build a winning DevOps culture and re-energize the IT organization. Here, we'll examine easy and cost-effective ways to increase collaboration, the application of Lean thinking to reduce IT waste, and what constitutes a comprehensive DevOps metrics program.